```
#include <iostream>
#include <utility>
#include <vector>
using namespace std;


template<typename K, typename V>
class my_map
{
public:
    my_map(){}
    ~my_map(){}
    void insert(K k,V v);
    V get(K k)const;
    V remove(K k);
private:
    vector<pair<K,V>> data;
};

template<typename K, typename V>
void my_map<K,V>::insert(K k, V v){
    for(auto& p: data)
        if(p.first == k){
            p.second = v;
            return;
        }
    data.push_back(pair<K,V>(k,v));
}

template<typename K, typename V>
V my_map<K,V>::get(K k)const{
    for(auto p: data)
        if(p.first == k)
            return p.second;
    return V();
}

template<typename K, typename V>
V my_map<K,V>::remove(K k){
  for(typename vector<pair<K,V>>::iterator it=data.begin();
      it != data.end(); it++)
      if(it->first == k){
          V ret = it->second;
          data.erase(it);
          return ret;
      }
  return V();
}

int main()
{
    my_map<string,int> numbers;
    numbers.insert("0", 0);
    numbers.insert("1", 1);
    numbers.insert("2", 2);
    numbers.insert("3", 3);
    numbers.insert("4", 4);
    numbers.insert("5", 5);
    cout << "get 3: " << numbers.get("3") << endl;
    numbers.remove("3");
    cout << "get 3: " << numbers.get("3") << endl;

    return 0;
}


--------------------------------------------------------

#include <iostream>
#include <map>
#include <set>

using namespace std;

struct person{
    string name, email;
    bool operator <(const person& other) const{
        return email < other.email;
    }
};

int main()
{

    set<person> sbook;
    sbook.insert(person{"Joe", "joe@liu.se"});
    sbook.insert(person{"Joe", "joe2@liu.se"});
    sbook.insert(person{"Joe_Bar", "joe@liu.se"});
    cout << "set content \n";
    for(person p: sbook)
       cout << p.name << " -> " << p.email << endl;
    return 0;
}


-----------------------------------------------------


#include <iostream>
#include <map>

using namespace std;

int main()
{
    multimap<string,int> m;
    m.insert({"0",0});
    m.insert({"1",1});
    m.insert({"2",2});
    m.insert({"3",3});

    for(multimap<string,int>::iterator it=m.begin(); it!=m.end(); it++)
        cout << it->first << " -> " << it->second << endl;
    cout << endl;
    m.insert({"1",-1});

    cout << "after change: \n";
    for(multimap<string,int>::iterator it=m.begin(); it!=m.end(); it++)
        cout << it->first << " -> " << it->second << endl;

    return 0;
}


-----------------------------------------------------------

#include <iostream>
#include <vector>
#include <list>
#include <algorithm>
#include <iterator>

using namespace std;

int main()
{
    vector<int> v{1,2,3,4,5};
    list<int> lst{-1,-2,-3};
    copy(v.begin(), v.end(), insert_iterator<list<int>>(lst,next(lst.begin())));
    for(int n: lst)
        cout << n << " ";
    cout << endl;

    return 0;
}


----------------------------------------------------

#include <iostream>
#include <iterator>

using namespace std;

int main()
{
    double val1, val2;
    cout << " insert two values: " << endl;

    istream_iterator<double> eos;
    istream_iterator<double> iit (cin);

    if(iit != eos)
        val1 = *iit;

    ++iit;

    if(iit != eos)
        val2 = *iit;

    cout << val1 << " * " << val2 << " = " << (val1 * val2) << endl;
    return 0;
```

```
}


----------------------------------------------------------

#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>
#include <numeric>
#include <string>

using namespace std;

int main()
{

    vector<string> v{"this", "is", "an", "example"};

    cout << "old content: ";
    for(auto& s: v)
        cout << " " << s;

    string concat= accumulate(vector<string>::iterator(v.begin()),
                              vector<string>::iterator(v.end()),
                              string());

    cout << "\nconcatenated as: " << concat << endl;
    cout << "\nnew content: ";
    for(auto& s: v)
        cout << " " << s;

    return 0;
}


---------------------------------------------------------

std::vector<int> v{1,2,3,4};
for(std::vector<int>::const_reverse_iterator it=v.crbegin(); it!=v.crend();++it)
{
  std::cout << ∗it << ’ ’;
}

std::cout<<std::endl;

// Switch crbegin with cend and crend with cbegin

for(vector<int>::const_iterator it=v.cbegin(); it!=v.cend(); −−it)
{
  std::cout << ∗(it−1) <<’ ’;
}
```